

Notable Examples in Isabelle/Pure

January 18, 2026

1 A simple formulation of First-Order Logic

The subsequent theory development illustrates single-sorted intuitionistic first-order logic with equality, formulated within the Pure framework.

```
theory First_Order_Logic
  imports Pure
begin
```

1.1 Abstract syntax

```
typeddecl i
typeddecl o

judgment Trueprop :: o ⇒ prop (⟨_⟩ 5)
```

1.2 Propositional logic

```
axiomatization false :: o (⟨⊥⟩)
  where falseE [elim]: ⊥ ⇒ A

  axiomatization imp :: o ⇒ o ⇒ o (infixr ⟨→⟩ 25)
    where impI [intro]: (A ⇒ B) ⇒ A ⇒ B
      and mp [dest]: A ⇒ B ⇒ A ⇒ B
```

```
  axiomatization conj :: o ⇒ o ⇒ o (infixr ⟨∧⟩ 35)
    where conjI [intro]: A ⇒ B ⇒ A ∧ B
      and conjD1: A ∧ B ⇒ A
      and conjD2: A ∧ B ⇒ B
```

```
theorem conjE [elim]:
  assumes A ∧ B
  obtains A and B
proof
  from ⟨A ∧ B⟩ show A
```

```

by (rule conjD1)
from ⟨A ∧ B⟩ show B
  by (rule conjD2)
qed

```

```

axiomatization disj :: o ⇒ o ⇒ o (infixr ⟨∨⟩ 30)
where disjE [elim]: A ∨ B ⇒ (A ⇒ C) ⇒ (B ⇒ C) ⇒ C
  and disjI1 [intro]: A ⇒ A ∨ B
  and disjI2 [intro]: B ⇒ A ∨ B

```

```

definition true :: o (⟨T⟩)
where T ≡ ⊥ → ⊥

```

```

theorem trueI [intro]: T
  unfolding true_def ..

```

```

definition not :: o ⇒ o (⟨¬⟩ [40] 40)
where ¬ A ≡ A → ⊥

```

```

theorem notI [intro]: (A ⇒ ⊥) ⇒ ¬ A
  unfolding not_def ..

```

```

theorem notE [elim]: ¬ A ⇒ A ⇒ B
  unfolding not_def
proof –
  assume A → ⊥ and A
  then have ⊥ ..
  then show B ..
qed

```

```

definition iff :: o ⇒ o ⇒ o (infixr ⟨↔⟩ 25)
where A ↔ B ≡ (A → B) ∧ (B → A)

```

```

theorem iffI [intro]:
  assumes A ⇒ B
  and B ⇒ A
  shows A ↔ B
  unfolding iff_def
proof
  from ⟨A ⇒ B⟩ show A → B ..
  from ⟨B ⇒ A⟩ show B → A ..
qed

```

```

theorem iff1 [elim]:
  assumes A ↔ B and A

```

shows B
proof –
 from $\langle A \leftrightarrow B \rangle$ have $(A \rightarrow B) \wedge (B \rightarrow A)$
 unfolding iff_def .
 then have $A \rightarrow B$..
 from *this* and $\langle A \rangle$ show B ..
qed

theorem iff2 [elim] :
 assumes $A \leftrightarrow B$ and B
 shows A
proof –
 from $\langle A \leftrightarrow B \rangle$ have $(A \rightarrow B) \wedge (B \rightarrow A)$
 unfolding iff_def .
 then have $B \rightarrow A$..
 from *this* and $\langle B \rangle$ show A ..
qed

1.3 Equality

axiomatization $\text{equal} :: i \Rightarrow i \Rightarrow o$ (**infixl** =: 50)
 where refl [intro] : $x = x$
 and $\text{subst} :: x = y \Rightarrow P x \Rightarrow P y$

theorem trans [trans] : $x = y \Rightarrow y = z \Rightarrow x = z$
 by (rule subst)

theorem sym [sym] : $x = y \Rightarrow y = x$
proof –
 assume $x = y$
 from *this* and refl show $y = x$
 by (rule subst)
qed

1.4 Quantifiers

axiomatization $\text{All} :: (i \Rightarrow o) \Rightarrow o$ (**binder** $\langle \forall \rangle$ 10)
 where allI [intro] : $(\bigwedge x. P x) \Rightarrow \forall x. P x$
 and allD [dest] : $\forall x. P x \Rightarrow P a$

axiomatization $\text{Ex} :: (i \Rightarrow o) \Rightarrow o$ (**binder** $\langle \exists \rangle$ 10)
 where exI [intro] : $P a \Rightarrow \exists x. P x$
 and exE [elim] : $\exists x. P x \Rightarrow (\bigwedge x. P x \Rightarrow C) \Rightarrow C$

lemma $(\exists x. P (f x)) \rightarrow (\exists y. P y)$
proof
 assume $\exists x. P (f x)$
 then obtain x where $P (f x)$..
 then show $\exists y. P y$..

```

qed

lemma ( $\exists x. \forall y. R x y$ )  $\longrightarrow$  ( $\forall y. \exists x. R x y$ )
proof
  assume  $\exists x. \forall y. R x y$ 
  then obtain  $x$  where  $\forall y. R x y ..$ 
  show  $\forall y. \exists x. R x y$ 
  proof
    fix  $y$ 
    from  $\forall y. R x y$  have  $R x y ..$ 
    then show  $\exists x. R x y ..$ 
  qed
qed

end

```

2 Foundations of HOL

```

theory Higher_Order_Logic
  imports Pure
begin

```

The following theory development illustrates the foundations of Higher-Order Logic. The “HOL” logic that is given here resembles [2] and its predecessor [1], but the order of axiomatizations and defined connectives has been adapted to modern presentations of λ -calculus and Constructive Type Theory. Thus it fits nicely to the underlying Natural Deduction framework of Isabelle/Pure and Isabelle/Isar.

3 HOL syntax within Pure

```

class type
default_sort type

typedecl o
instance o :: type ..
instance fun :: (type, type) type ..

judgment Trueprop :: o  $\Rightarrow$  prop ( $\hookrightarrow$  5)

```

4 Minimal logic (axiomatization)

```

axiomatization imp :: o  $\Rightarrow$  o  $\Rightarrow$  o (infixr  $\hookrightarrow$  25)
  where impI [intro]:  $(A \Rightarrow B) \Rightarrow A \rightarrow B$ 
    and impE [dest, trans]:  $A \rightarrow B \Rightarrow A \Rightarrow B$ 

axiomatization All :: ('a  $\Rightarrow$  o)  $\Rightarrow$  o (binder  $\forall$  10)

```

where *allI* [intro]: $(\bigwedge x. P x) \implies \forall x. P x$
and *allE* [dest]: $\forall x. P x \implies P a$

lemma *atomize_imp* [atomize]: $(A \implies B) \equiv \text{Trueprop } (A \longrightarrow B)$
by standard (fact *impI*, fact *impE*)

lemma *atomize_all* [atomize]: $(\bigwedge x. P x) \equiv \text{Trueprop } (\forall x. P x)$
by standard (fact *allI*, fact *allE*)

4.0.1 Derived connectives

definition *False* :: *o*
where *False* $\equiv \forall A. A$

lemma *FalseE* [elim]:
assumes *False*
shows *A*
proof –
 from *False* have $\forall A. A$ by (simp only: *False_def*)
 then show *A* ..
qed

definition *True* :: *o*
where *True* $\equiv \text{False} \longrightarrow \text{False}$

lemma *TrueI* [intro]: *True*
unfolding *True_def* ..

definition *not* :: *o* \Rightarrow *o* (\neg [40] 40)
where *not* $\equiv \lambda A. A \longrightarrow \text{False}$

lemma *notI* [intro]:
assumes *A* $\implies \text{False}$
shows $\neg A$
using *assms* **unfolding** *not_def* ..

lemma *notE* [elim]:
assumes $\neg A$ **and** *A*
shows *B*
proof –
 from $\neg A$ have *A* $\longrightarrow \text{False}$ by (simp only: *not_def*)
 from this and *A* have *False* ..
 then show *B* ..
qed

lemma *notE'*: $A \implies \neg A \implies B$
by (rule *notE*)

```
lemmas contradiction = noteE noteE' — proof by contradiction in any order
```

```
definition conj :: o ⇒ o ⇒ o (infixr ∧ 35)
  where A ∧ B ≡ ∀ C. (A → B → C) → C
```

```
lemma conjI [intro]:
  assumes A and B
  shows A ∧ B
  unfolding conj_def
proof
  fix C
  show (A → B → C) → C
proof
  assume A → B → C
  also note ⟨A⟩
  also note ⟨B⟩
  finally show C .
qed
qed
```

```
lemma conjE [elim]:
  assumes A ∧ B
  obtains A and B
proof
  from ⟨A ∧ B⟩ have *: (A → B → C) → C for C
  unfolding conj_def ..
  show A
  proof -
    note * [of A]
    also have A → B → A
    proof
      assume A
      then show B → A ..
    qed
    finally show ?thesis .
  qed
  show B
  proof -
    note * [of B]
    also have A → B → B
    proof
      show B → B ..
    qed
    finally show ?thesis .
  qed
qed
```

```

definition disj ::  $o \Rightarrow o \Rightarrow o$  (infixr  $\langle\vee\rangle$  30)
  where  $A \vee B \equiv \forall C. (A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C$ 

lemma disjI1 [intro]:
  assumes  $A$ 
  shows  $A \vee B$ 
  unfolding disj_def
proof
  fix  $C$ 
  show  $(A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C$ 
  proof
    assume  $A \rightarrow C$ 
    from this and  $\langle A \rangle$  have  $C$  ..
    then show  $(B \rightarrow C) \rightarrow C$  ..
  qed
qed

lemma disjI2 [intro]:
  assumes  $B$ 
  shows  $A \vee B$ 
  unfolding disj_def
proof
  fix  $C$ 
  show  $(A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C$ 
  proof
    show  $(B \rightarrow C) \rightarrow C$ 
    proof
      assume  $B \rightarrow C$ 
      from this and  $\langle B \rangle$  show  $C$  ..
    qed
  qed
qed

lemma disjE [elim]:
  assumes  $A \vee B$ 
  obtains (a)  $A \mid (b) B$ 
proof –
  from  $\langle A \vee B \rangle$  have  $(A \rightarrow \text{thesis}) \rightarrow (B \rightarrow \text{thesis}) \rightarrow \text{thesis}$ 
  unfolding disj_def ..
  also have  $A \rightarrow \text{thesis}$ 
  proof
    assume  $A$ 
    then show  $\text{thesis}$  by (rule a)
  qed
  also have  $B \rightarrow \text{thesis}$ 
  proof
    assume  $B$ 
    then show  $\text{thesis}$  by (rule b)

```

```

qed
  finally show thesis .
qed

definition Ex :: ('a  $\Rightarrow$  o)  $\Rightarrow$  o (binder  $\langle \exists \rangle$  10)
  where  $\exists x. P x \equiv \forall C. (\forall x. P x \rightarrow C) \rightarrow C$ 

lemma exI [intro]: P a  $\Rightarrow$   $\exists x. P x$ 
  unfolding Ex_def
proof
  fix C
  assume P a
  show  $(\forall x. P x \rightarrow C) \rightarrow C$ 
proof
  assume  $\forall x. P x \rightarrow C$ 
  then have P a  $\rightarrow C$  ..
  from this and  $\langle P a \rangle$  show C ..
qed
qed

lemma exE [elim]:
  assumes  $\exists x. P x$ 
  obtains (that) x where P x
proof -
  from  $\langle \exists x. P x \rangle$  have  $(\forall x. P x \rightarrow \text{thesis}) \rightarrow \text{thesis}$ 
  unfolding Ex_def ..
  also have  $\forall x. P x \rightarrow \text{thesis}$ 
proof
  fix x
  show P x  $\rightarrow \text{thesis}$ 
proof
  assume P x
  then show thesis by (rule that)
qed
qed
  finally show thesis .
qed

```

4.0.2 Extensional equality

```

axiomatization equal :: 'a  $\Rightarrow$  'a  $\Rightarrow$  o (infixl  $\langle = \rangle$  50)
  where refl [intro]: x = x
        and subst: x = y  $\Rightarrow$  P x  $\Rightarrow$  P y

abbreviation not_equal :: 'a  $\Rightarrow$  'a  $\Rightarrow$  o (infixl  $\langle \neq \rangle$  50)
  where x  $\neq$  y  $\equiv$   $\neg (x = y)$ 

abbreviation iff :: o  $\Rightarrow$  o  $\Rightarrow$  o (infixr  $\langle \leftrightarrow \rangle$  25)

```

where $A \longleftrightarrow B \equiv A = B$

axiomatization

where ext [intro]: $(\bigwedge x. f x = g x) \implies f = g$
and iff [intro]: $(A \implies B) \implies (B \implies A) \implies A \longleftrightarrow B$
for $f g :: 'a \Rightarrow 'b$

lemma sym [sym]: $y = x$ **if** $x = y$
using that **by** (rule subst) (rule refl)

lemma [trans]: $x = y \implies P y \implies P x$
by (rule subst) (rule sym)

lemma [trans]: $P x \implies x = y \implies P y$
by (rule subst)

lemma arg_cong : $f x = f y$ **if** $x = y$
using that **by** (rule subst) (rule refl)

lemma fun_cong : $f x = g x$ **if** $f = g$
using that **by** (rule subst) (rule refl)

lemma trans [trans]: $x = y \implies y = z \implies x = z$
by (rule subst)

lemma iff1 [elim]: $A \longleftrightarrow B \implies A \implies B$
by (rule subst)

lemma iff2 [elim]: $A \longleftrightarrow B \implies B \implies A$
by (rule subst) (rule sym)

4.1 Cantor's Theorem

Cantor's Theorem states that there is no surjection from a set to its powerset. The subsequent formulation uses elementary λ -calculus and predicate logic, with standard introduction and elimination rules.

lemma iff_contradiction :
assumes $*: \neg A \longleftrightarrow A$
shows C
proof (rule noteE)
show $\neg A$
proof
assume A
with $*$ have $\neg A ..$
from this and $\langle A \rangle$ show *False* ..
qed
with $*$ show $A ..$
qed

```

theorem Cantor:  $\neg (\exists f :: 'a \Rightarrow 'a \Rightarrow o. \forall A. \exists x. A = f x)$ 
proof
  assume  $\exists f :: 'a \Rightarrow 'a \Rightarrow o. \forall A. \exists x. A = f x$ 
  then obtain  $f :: 'a \Rightarrow 'a \Rightarrow o$  where  $*: \forall A. \exists x. A = f x ..$ 
  let  $?D = \lambda x. \neg f x x$ 
  from  $*$  have  $\exists x. ?D = f x ..$ 
  then obtain  $a$  where  $?D = f a ..$ 
  then have  $?D a \longleftrightarrow f a a$  using refl by (rule subst)
  then have  $\neg f a a \longleftrightarrow f a a ..$ 
  then show False by (rule iff_contradiction)
qed

```

4.2 Characterization of Classical Logic

The subsequent rules of classical reasoning are all equivalent.

```

locale classical =
  assumes classical:  $(\neg A \Rightarrow A) \Rightarrow A$ 
  — predicate definition and hypothetical context
begin

lemma classical_contradiction:
  assumes  $\neg A \Rightarrow False$ 
  shows  $A$ 
  proof (rule classical)
    assume  $\neg A$ 
    then have False by (rule assms)
    then show  $A ..$ 
  qed

lemma double_negation:
  assumes  $\neg \neg A$ 
  shows  $A$ 
  proof (rule classical_contradiction)
    assume  $\neg A$ 
    with  $\neg \neg A$  show False by (rule contradiction)
  qed

lemma tertium_non_datur:  $A \vee \neg A$ 
  proof (rule double_negation)
    show  $\neg \neg (A \vee \neg A)$ 
    proof
      assume  $\neg (A \vee \neg A)$ 
      have  $\neg A$ 
      proof
        assume  $A$  then have  $A \vee \neg A ..$ 
        with  $\neg (A \vee \neg A)$  show False by (rule contradiction)
      qed
      then have  $A \vee \neg A ..$ 
      with  $\neg (A \vee \neg A)$  show False by (rule contradiction)
    qed
  
```

```

qed
qed

lemma classical_cases:
  obtains  $A \mid \neg A$ 
  using tertium_non_datur
proof
  assume  $A$ 
  then show thesis ..
next
  assume  $\neg A$ 
  then show thesis ..
qed

end

lemma classical_if_cases: classical
  if cases:  $\bigwedge A \ C. (A \Rightarrow C) \Rightarrow (\neg A \Rightarrow C) \Rightarrow C$ 
proof
  fix  $A$ 
  assume *:  $\neg A \Rightarrow A$ 
  show  $A$ 
  proof (rule cases)
    assume  $A$ 
    then show  $A$  .
  next
    assume  $\neg A$ 
    then show  $A$  by (rule *)
  qed
qed

```

5 Peirce's Law

Peirce's Law is another characterization of classical reasoning. Its statement only requires implication.

```

theorem (in classical) Peirce's_Law:  $((A \rightarrow B) \rightarrow A) \rightarrow A$ 
proof
  assume *:  $(A \rightarrow B) \rightarrow A$ 
  show  $A$ 
  proof (rule classical)
    assume  $\neg A$ 
    have  $A \rightarrow B$ 
    proof
      assume  $A$ 
      with  $\neg A$  show  $B$  by (rule contradiction)
    qed
    with * show  $A$  ..
  qed

```

qed

6 Hilbert's choice operator (axiomatization)

axiomatization $Eps :: ('a \Rightarrow o) \Rightarrow 'a$
 where $someI: P x \implies P (Eps P)$

syntax $_Eps :: pttrn \Rightarrow o \Rightarrow 'a$ ((*indent=3 notation=binder SOME*)
 $_./ _)$ [0, 10] 10)
syntax_consts $_Eps \Leftarrow Eps$
translations $SOME x. P \Leftarrow CONST Eps (\lambda x. P)$

It follows a derivation of the classical law of tertium-non-datur by means of Hilbert's choice operator (due to Berghofer, Beeson, Harrison, based on a proof by Diaconescu).

theorem *Diaconescu*: $A \vee \neg A$

proof –

let $?P = \lambda x. (A \wedge x) \vee \neg x$
 let $?Q = \lambda x. (A \wedge \neg x) \vee x$

have $a: ?P (Eps ?P)$
 proof (*rule someI*)
 have $\neg False ..$
 then show $?P False ..$
 qed
 have $b: ?Q (Eps ?Q)$
 proof (*rule someI*)
 have $True ..$
 then show $?Q True ..$
 qed

from a **show** $?thesis$
proof

assume $A \wedge Eps ?P$
 then have $A ..$
 then show $?thesis ..$

next

assume $\neg Eps ?P$
 from b **show** $?thesis$
proof

assume $A \wedge \neg Eps ?Q$
 then have $A ..$
 then show $?thesis ..$

next

assume $Eps ?Q$
 have $neq: ?P \neq ?Q$
proof

```

assume ?P = ?Q
then have Eps ?P  $\longleftrightarrow$  Eps ?Q by (rule arg_cong)
also note <Eps ?Q>
finally have Eps ?P .
with < $\neg$  Eps ?P> show False by (rule contradiction)
qed
have  $\neg A$ 
proof
  assume A
  have ?P = ?Q
  proof (rule ext)
    show ?P x  $\longleftrightarrow$  ?Q x for x
  proof
    assume ?P x
    then show ?Q x
  proof
    assume  $\neg x$ 
    with <A> have A  $\wedge$   $\neg x$  ..
    then show ?thesis ..
  next
    assume A  $\wedge$  x
    then have x ..
    then show ?thesis ..
  qed
next
  assume ?Q x
  then show ?P x
  proof
    assume A  $\wedge$   $\neg x$ 
    then have  $\neg x$  ..
    then show ?thesis ..
  next
    assume x
    with <A> have A  $\wedge$  x ..
    then show ?thesis ..
  qed
  qed
qed
with neq show False by (rule contradiction)
qed
then show ?thesis ..
qed
qed
qed

```

This means, the hypothetical predicate *classical* always holds unconditionally (with all consequences).

```

interpretation classical
proof (rule classical_if_cases)

```

```

fix A C
assume *: A  $\implies$  C
  and **:  $\neg A \implies C$ 
from Diaconescu [of A] show C
proof
  assume A
  then show C by (rule *)
next
  assume  $\neg A$ 
  then show C by (rule **)
qed
qed

thm classical
classical_contradiction
double_negation
tertium_non_datur
classical_cases
Peirce's_Law

end

```

References

- [1] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
- [2] M. J. C. Gordon. HOL: A machine oriented formulation of higher order logic. Technical Report 68, University of Cambridge Computer Laboratory, 1985.